Fuzzy Inverse Kinematic Mapping: Rule Generation, Efficiency, and Implementation

Yangsheng Xu **Michael C. Nechyba**

CMU-RI-TR-93-02

The Robotics Institute
Carnegie Mellon University
Pittsburgh , Pennsylvania 15213

January 1993

This material is based upon work partially supported under a National Science Foundation Graduate Research Fellowship.

REPORT DOCUMENTATION PAGE	*Form Approved* OMB No 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE January 1993	3. REPORT TYPE AND DATES COVERED technical

4. TITLE AND SUBTITLE

Fuzzy Inverse Kinematic Mapping: Rule Generation, Efficiency, and Implementation

5. FUNDING NUMBERS

6. AUTHOR(S)

Yangsheng Xu, Michael C. Nechyba

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER

CMU-RI-TR-93-02

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release;
Distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Inverse kinematics is computationally expensive and can result in significant control delays in real time. For a redundant robot, additional computations are required for the inverse kinematic solution through optimization schemes. Based on the fact that humans do not compute exact inverse kinematics, but can do precise positioning from heuristics, we developed an inverse kinematic mapping through fuzzy logic. The implementation of the proposed scheme has demonstrated that it is feasible for both redundant and nonredundant cases, and that it is very computationally efficient. The result provides sufficient precision, and transient tracking error can be controlled based on a fuzzy adaptive scheme proposed in this paper. This paper discusses (1) the automatic generation of the Fuzzy Inverse Kinematic Mapping (FIKM) from specification of the DH parameters, (2) the efficiency of the scheme in comparison to conventional approaches, and (3) the implementation results for both redundant and nonredundant robots.

14. SUBJECT TERMS	15. NUMBER OF PAGES 28 pp
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT **unlimited**	18. SECURITY CLASSIFICATION OF THIS PAGE **unlimited**	19. SECURITY CLASSIFICATION OF ABSTRACT **unlimited**	20. LIMITATION OF ABSTRACT **unlimited**

Contents

List of Figures

List of Tables

Abstract

Inverse kinematics is computationally expensive and can result in significant control delays in real time. For a redundant robot, additional computations are required for the inverse kinematic solution through optimization schemes. Based on the fact that humans do not compute exact inverse kinematics, but can do precise positioning from heuristics, we developed an inverse kinematic mapping through fuzzy logic. The implementation of the proposed scheme has demonstrated that it is feasible for both redundant and nonredundant cases, and that it is very computationally efficient. The result provides sufficient precision, and transient tracking error can be controlled based on a fuzzy adaptive scheme proposed in this paper. This paper discusses (1) the automatic generation of the Fuzzy Inverse Kinematic Mapping (FIKM) from specification of the DH parameters, (2) the efficiency of the scheme in comparison to conventional approaches, and (3) the implementation results for both redundant and nonredundant robots.

1 Introduction

One of the major problems of robot manipulator control today is that of calculating inverse kinematics in real time. Calculating inverse kinematics is computationally expensive and generally consumes a large percentage of time in the real-time control of robot manipulators.

The problem of inverse kinematics may be summarized as follows: Given the 6x1 position/orientation vector *r* of the end-effector in Cartesian space, calculate the *n*x1 vector of joint angles Θ required to place the end-effector at the desired position and orientation. Here, *n* represents the number of degrees of freedom (DOF) of the manipulator. In general, inverse kinematics does not result in one-to-one mapping between Cartesian and joint space, and closed-form solutions to the inverse kinematic problem exist only for a very small class of kinematically simple manipulators [2].

In the case of redundant manipulators and nonredundant manipulators in singular configurations, the problem is compounded by the fact that throughout the workspace of the manipulator, multiple solutions (perhaps even an infinite number of solutions) exist. The inverse kinematics of redundant manipulators therefore requires that a choice be made among the set of all possible solutions. Arriving at such a decision through some optimization scheme is difficult and the time-consuming computations can result in significant control delays.

Humans do not, however, have to calculate exact inverse kinematics every time we move an arm or a leg. Experience and knowledge, rather than complex computations, allow humans to effectively move with ease. In this paper, we propose to characterize this human knowledge by proposing a general method of computing the inverse kinematics for an arbitrary *n*-DOF manipulator through a fuzzy logic approach. The method applies equally well for redundant and nonredundant manipulators, is computationally efficient, and robust at or near singular configurations. The scheme has been implemented in the real-time control of a teleoperated space robot [7], and the results have shown that the scheme is very efficient, especially in teleoperation.

In this paper, we first present an algorithm which automatically generates the fuzzy model for an arbitrary manipulator based only on the Denavit-Hartenberg (DH) parameters [2]. Second, we analyze the generated fuzzy model characteristics and present a very efficient method of indirectly calculating the fuzzy model output. Third, we present simulation results for two redundant and one nonredundant manipulator. Fourth, we analyze the computational efficiency of our method and compare it to other current methods for computing inverse kinematics.

2 Fuzzy Model Generation

2.1 Overview

As shown in Figure 1, our fuzzy inverse kinematic mapping (FIKM) takes as input the actual and desired locations of the end-effector, and the current joint variable values. From these inputs, the fuzzy controller generates as output the necessary trajectories for the joint variables, so that the actual and desired end-effector locations converge to zero steady-state error.

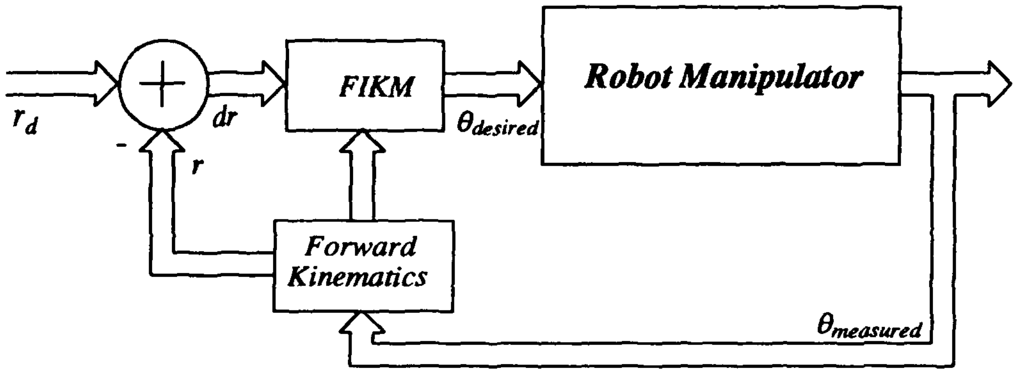


Figure 1: The overall signal flow for the fuzzy controller.

The Jacobian matrix $J(\Theta)$ relates the differential Cartesian rates dr to the differential joint rates $d\theta$, such that

$$dr = J(\Theta)\,d\theta \qquad \text{(Eq. 1)}$$

Essentially, we want to solve the inverse problem to (Eq. 1), namely,

$$d\theta = J^{-1}(\Theta)\,dr \qquad \text{(Eq. 2)}$$

There are many reasons why we cannot solve (Eq. 2) analytically, however. First, $J^{-1}(\Theta)$ exists only when $n = 6$, and therefore is not suited for redundant manipulators. Second, even when we can solve for $J^{-1}(\Theta)$, the solution will degenerate at and near singularities. Third, the computations involved in inverting a 6x6 matrix in real time are time consuming. Although there are some algorithms available, as in [5] and [6], the computations are still complex. Therefore, we propose a fuzzy logic approach to solving the problem. Figure 2 outlines the overall algorithm we use to generate the fuzzy mapping automatically, with only the DH parameters as input to the algorithm.

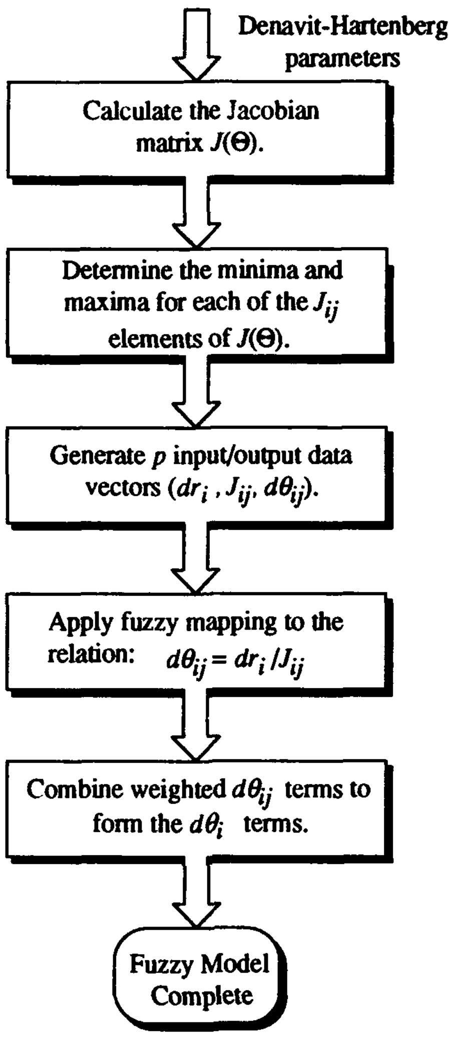
Consider each J_{ij} term in the Jacobian separately along with dr_i, the *ith* component of the dr vector. We define a new variable $d\theta_{ij}$ which relates dr_i and J_{ij},

$$J_{ij}d\theta_{ij} \approx dr_i \qquad \text{(Eq. 3)}$$

Therefore, $d\theta_{ij}$ relates how much $d\theta_j$ contributes to dr_i. This relationship gives a good understanding of which joints will contribute more to reducing dr_i and which ones will contribute less. Thus, with proper scaling of each of the $d\theta_{ij}$'s the fuzzy mapping can arrive at an intelligent set of joint angles that will drive the end-effector to the desired position. The function that we will actually apply the fuzzy mapping to is given by,

$$d\theta_{ij} \approx \frac{dr_i}{J_{ij}} \qquad \text{(Eq. 4)}$$

The following sections discuss in detail each of the steps described briefly in Figure 2.



Denavit-Hartenberg parameters
Calculate the Jacobian matrix $J(\Theta)$.
Determine the minima and maxima for each of the J_{ij} elements of $J(\Theta)$.
Generate p input/output data vectors $(dr_i, J_{ij}, d\theta_{ij})$.
Apply fuzzy mapping to the relation: $d\theta_{ij} = dr_i / J_{ij}$
Combine weighted $d\theta_{ij}$ terms to form the $d\theta_i$ terms.
Fuzzy Model Complete

Figure 2: The flowchart illustrates the overall algorithm that was used to generate the fuzzy model, given .only the DH parameters as input.

2.2 Jacobian Calculation

There are many ways of calculating the Jacobian, of which a computationally efficient one is the most attractive. We briefly describe one such efficient method below [6]. First, divide the Jacobian matrix into $2n$ submatrices:

$$J = \begin{bmatrix} J_{11} & \cdots & J_{1n} \\ J_{21} & \cdots & J_{2n} \end{bmatrix} \qquad \text{(Eq. 5)}$$

Each of the submatrices in (Eq. 5) are 3 x 1 column vectors.

The J_{1i}, $i \in \{1, \ldots, n\}$, vectors may be calculated recursively from the base towards the end-effector, by applying the following iterative scheme:

$$\begin{aligned} \tilde{J}_{10} &= \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T & & \\ \tilde{J}_{1i} &= \tilde{J}_{1(i-1)} + {}^0R_i{}^iP_i & i &\in \{1, 2, \ldots, n-1\} \\ J_{1i} &= -J_{2i} \times \tilde{J}_{1(i-1)} & i &\in \{1, 2, \ldots, n\} \end{aligned} \qquad \text{(Eq. 6)}$$

where ${}^0R_i$ denotes the rotation transformation from link i to link 0. The J_{2i}, $i \in \{1, \ldots, n\}$, vectors may be calculated recursively from the base toward the end-effector, by applying the iterative scheme,

$$J_{2i} = {}^0R_{i-1} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \qquad \text{(Eq. 7)}$$

The resultant $J(\Theta)$ is expressed in the base frame of the manipulator.

2.3 Range Determination

In order to minimize the inference error of the fuzzy model, we want to fuzzify relationship (Eq. 4) over the full range of values that J_{ij} may assume. Therefore it is useful to determine, before the fuzzy mapping, the range for each element J_{ij}, $i \in \{1, \ldots, 6\}$, $j \in \{1, \ldots, n\}$, in $J(\Theta)$. Each J_{ij} will be of the form,

$$J_{ij} = l_1 f_1(\theta_1, \ldots, \theta_n) + l_2 f_2(\theta_1, \ldots, \theta_n) + \ldots + l_k f(\theta_1, \ldots, \theta_n) + d_1 f_{k+1}(\theta_1, \ldots, \theta_n) + \ldots + d_m f_{k+m}(\theta_1, \ldots, \theta_n) \qquad \text{(Eq. 8)}$$

where each l_i, $i \in \{1, \ldots, k\}$, is a constant, each d_i, $i \in \{1, \ldots, m \le n\}$ is an offset distance in the DH parameters which may be variable, and each of the $f_i(\theta_1, \ldots, \theta_n)$, $i \in \{1, \ldots, k+m\}$, is a product of sine and cosine terms. Note, of course, that any (or all) of the coefficients d_i and l_i may be equal to 0 or 1.

The maximum and minimum values for the cosine and sine functions are $\forall x \in \{-\infty, \infty\}$,

$$\begin{aligned} \sin(x)_{min} &= -1 \\ \cos(x)_{min} &= -1 \\ \sin(x)_{max} &= 1 \\ \cos(x)_{max} &= 1 \end{aligned} \qquad \text{(Eq. 9)}$$

Therefore, it follows that for $0 \le \theta_i < 2\pi$, $i \in \{1, \ldots, n\}$, the maximum and minimum values for each $f_i(\theta_1, \ldots, \theta_n)$ are,

$$\begin{aligned} f_i(\theta_1, \ldots, \theta_n)_{min} &= -1 \\ f_i(\theta_1, \ldots, \theta_n)_{max} &= 1 \end{aligned} \qquad \text{(Eq. 10)}$$

In addition, for a given manipulator, we also know the range of allowable values for each variable d_i *a priori*. Then, the minimum and maximum values for each J_{ij}, are,

$$J_{ijmin} = -\left(\sum_{p=1}^{n} |l_p| + \sum_{q=1}^{m} |d_q|_{max}\right) \qquad \text{(Eq. 11)}$$
$$J_{ijmax} = |J_{ijmin}|$$

so that $J_{ijmin} \le J_{ij} \le J_{ijmax}$ for all possible θ_i and d_i.

2.4 Generation of Input/Output Data Table

In preparation for generating the fuzzy model, we require that a table of input/output data be generated for each $d\theta_{ij}$. Each input/output vector in the table must be of the form $(dr_i, J_{ij}, d\theta_{ij})$, where, of course, dr_i and J_{ij} are considered the inputs and $d\theta_{ij}$ is the output of the fuzzy model.

For the fuzzy mapping presented in this paper, we generated the input/output data by computing a table of vectors of the form,

$$^{P}(J_{ij} \times d\theta_{ij}, J_{ij}, d\theta_{ij}) \qquad \text{(Eq. 12)}$$

where J_{ij} was swept from J_{ijmin} to J_{ijmax} in $J_{ijmax}/1000$ increments, and $d\theta_{ij}$ was swept from $-0.1/J_{ijmax}$ to $0.1/J_{ijmax}$ in $0.04/J_{ijmax}$ increments. Therefore, the ranges for the input variables were,

$$-0.1 \le dr_i \le 0.1 \qquad \text{(Eq. 13)}$$
$$J_{ijmin} \le J_{ij} \le J_{ijmax}$$

The range for dr_i indicates our expectation that the end-effector will move less than 10 cm (0.1 m) in each direction per control cycle. We later see that we can expand the range for dr_i without any loss in accuracy.

This approach for generating the table avoids the problem of the singularity at $J_{ij} = 0$, had the input/output data been generated by vectors of the form,

$$\left(dr_i, J_{ij}, \frac{dr_i}{J_{ij}}\right) \qquad \text{(Eq. 14)}$$

Repeated points in the input/output table were eliminated before the table was used to generate the fuzzy model.

2.5 Generation of Fuzzy Model

Figure 3 outlines the algorithm that was used to generate the rule base and membership functions for the $d\theta_{ij}$'s. A similar approach for generating the fuzzy model can be found in [1]. The algorithm takes as input the table of input/output data generated in the previous section.

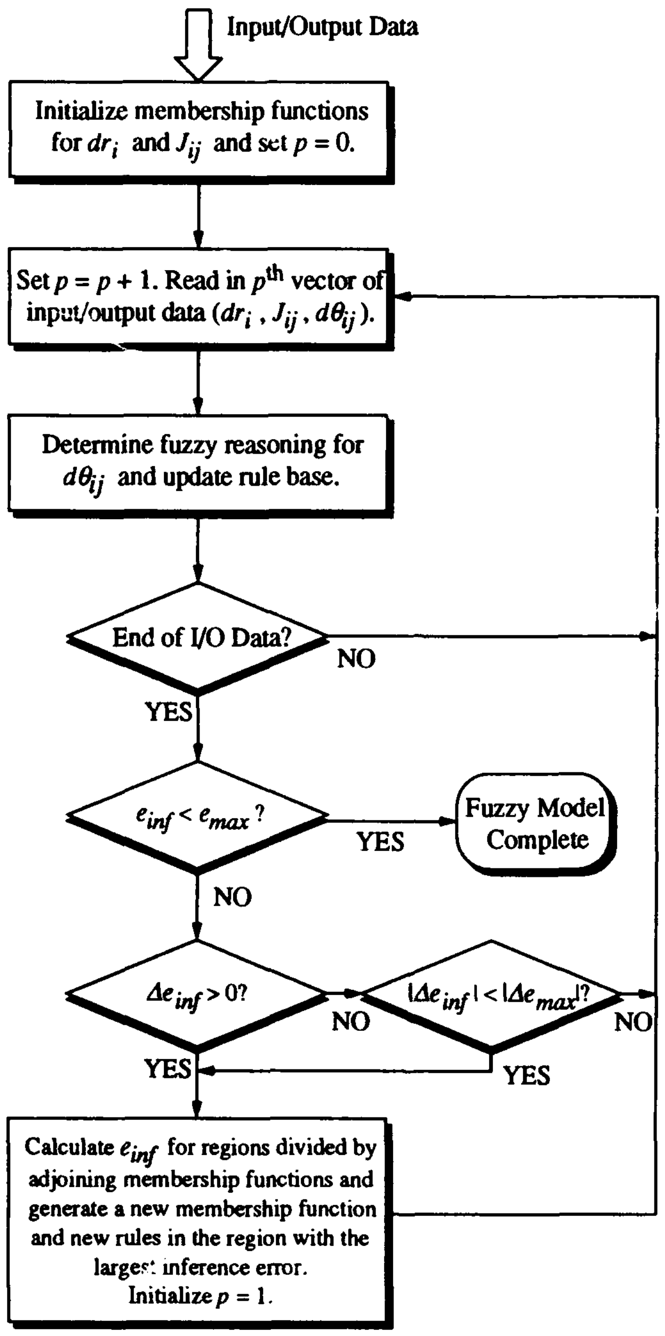

Input/Output Data
Initialize membership functions for dr_i and J_{ij} and set $p = 0$.
Set $p = p + 1$. Read in p^{th} vector of input/output data (dr_i, J_{ij}, $d\theta_{ij}$).
Determine fuzzy reasoning for $d\theta_{ij}$ and update rule base.
End of I/O Data?
NO
YES
$e_{inf} < e_{max}$?
YES
Fuzzy Model Complete
NO
$\Delta e_{inf} > 0$?
NO
$|\Delta e_{inf}| < |\Delta e_{max}|$?
NO
YES
YES
Calculate e_{inf} for regions divided by adjoining membership functions and generate a new membership function and new rules in the region with the largest inference error. Initialize $p = 1$.

Figure 3: Detailed algorithm for generating fuzzy mapping for $d\theta_{ij}$.

Initially, we generate three evenly spaced membership function per input variable, i.e., dr_i and J_{ij}, for the ranges that were determined previously. Figure 4 shows the initial arrangement of the membership functions for J_{ij}. Note that the sum of the membership functions at each value for J_{ij} add up 1. The initialized membership functions for dr_i are equivalent except for the range of dr_i.

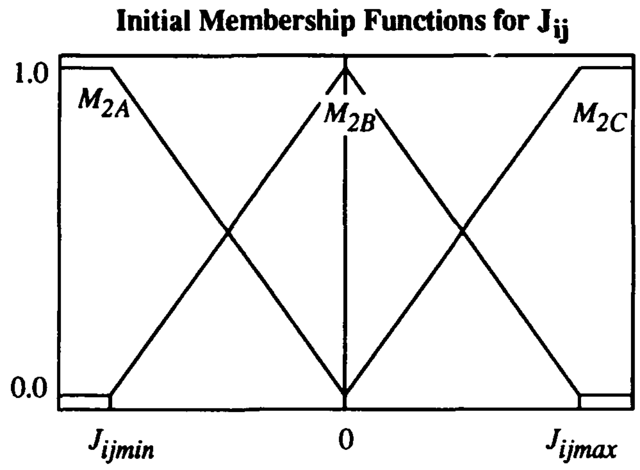


Figure 4: These were the initial membership functions that were used for J_{ij}. Similar three membership functions were used for dr_i.

Let N_{dri} = current number of membership functions for dr_i, and N_{Jij} = current number of membership functions for $d\theta_{ij}$. Denote M_{1x}, $x \in \{A, B, \ldots\}$, as a membership function of dr_i and denote M_{2y}, $y \in \{A, B, \ldots\}$, as a membership function of J_{ij}. Also, let $M_{1,m} = M_{1x}$, $m \in \{1, 2, \ldots, N_{dri}\}$, and $M_{2,m} = M_{2y}$, $m \in \{1, 2, \ldots, N_{Jij}\}$, where m denotes the position of the mth letter in the alphabet.

The initial rule base for the fuzzy control model takes the following form where R^k denotes the kth rule:

R^0: If dr_i is M_{1A} and J_{ij} is M_{2A} then $d\theta_{ij}$ is w_0.

R^1: If dr_i is M_{1A} and J_{ij} is M_{2B} then $d\theta_{ij}$ is w_1. (Eq. 15)

:

R^{N-1}: If dr_i is M_{1C} and J_{ij} is M_{2C} then $d\theta_{ij}$ is w_{N-1}.

Here, N = total number of rules which is equal to 9 at initialization. In general,

$$N = N_{dri} \times N_{Jij} \quad \text{(Eq. 16)}$$

$$R^{(q-1)N_{Jij}+r-1}: \text{If } dr_i \text{ is } M_{1,q} \text{ and } J_{ij} \text{ is } M_{2,r} \text{ then } d\theta_{ij} \text{ is } w_{(q-1)N_{Jij}+r-1} \quad \text{(Eq. 17)}$$

We further initialize the real numbers of the consequent part w_k, $k \in \{0, \ldots, N-1\}$, to 0, and the maximum inference error e_{max} to 0.0013. This number was arrived at iteratively after tries with various different stop values. When the inference error $e_{inf} < e_{max}$, then the fuzzy model is complete to desired specifications for that $d\theta_{ij}$.

Furthermore, we set a minimum threshold value $\Delta e_{max} = 5\times10^{-6}$ for the change in the inference error in consecutive iterations of the algorithm. This is used to decide whether or not to further reduce the inference error without generating new rules. If $\Delta e_{inf} > 0$ or $|\Delta e_{max}| < |\Delta e_{inf}|$ then a new membership function and new rules will be generated; if not, then further reduction of e_{inf} will be accomplished by adjusting w_k with repeated reading of the input/output data.

For each new input/output data vector p, we first calculate the truth value μ_k for each of the rules for that particular combination of inputs. For example,

$$\begin{aligned}
\mu_0 &= M_{1A}(dr_i)_p M_{2A}(J_{ij})_p \\
\mu_1 &= M_{1A}(dr_i)_p M_{2B}(J_{ij})_p \\
\mu_2 &= M_{1A}(dr_i)_p M_{2C}(J_{ij})_p \\
\mu_3 &= M_{1B}(dr_i)_p M_{2A}(J_{ij})_p \\
\mu_4 &= etc\ldots
\end{aligned} \qquad \text{(Eq. 18)}$$

Second, the output $(d\theta_{ij})_p^*$ of the fuzzy model is calculated by,

$$(d\theta_{ij})_p^* = \sum_{k=0}^{N-1} \mu_k \times w_k \qquad \text{(Eq. 19)}$$

The real numbers of the consequent part w_k are updated by,

$$w_k^{new} = w_k^{old} - c_w \mu_k [(d\theta_{ij})^*_p - (d\theta_{ij})_p] \qquad \text{(Eq. 20)}$$

where $c_w = 0.6$. This value was determined experimentally, and increases the speed of convergence to the smallest possible inference error with the least number of additional rules and membership functions being generated.

Once, the end of the input/output data has been reached, the average inference error is calculated by comparing the fuzzy-model output with the actual output for every input/output data vector,

$$e_{inf} = \frac{1}{p_{max}} \sum_{p=1}^{p_{max}} |(d\theta_{ij})^*_p - (d\theta_{ij})_p| \qquad \text{(Eq. 21)}$$

Furthermore, the change in the inference error from the previous cycle is also calculated,

$$\Delta e_{inf} = e_{inf}^{new} - e_{inf}^{old} \qquad \text{(Eq. 22)}$$

As can be observed from Figure 3, if $e_{inf} < e_{max} = 0.0013$, then the fuzzy model is complete. If the negative change in the inference error is still significant without generating new membership functions, the w_k are refined more by reprocessing the input/output data table. If, however, the Δe_{inf} is positive or negligibly negative, then new membership functions have to be generated in order to further reduce the inference error. Figure 5 shows by example how new membership functions are generated and how consequently, new rules are formed. First, the dr_i-J_{ij} plane is divided into $(N_{dri} - 1)$ x $(N_{Jij} - 1)$ regions. The region borders are generated where two adjoining membership functions meet. The top part of Figure 5, for example, has region R_{11} shaded. Second, the inference error is calculated for each region. This is done by summing up only those terms in (Eq. 21) for the p where $(dr_i)_p$ and $(J_{ij})_p$ fall within that specific region. We then select the region R_{max} with the greatest inference error where a new membership function is to be generated. Figure5 assumes that region $R_{11} = R_{max}$.

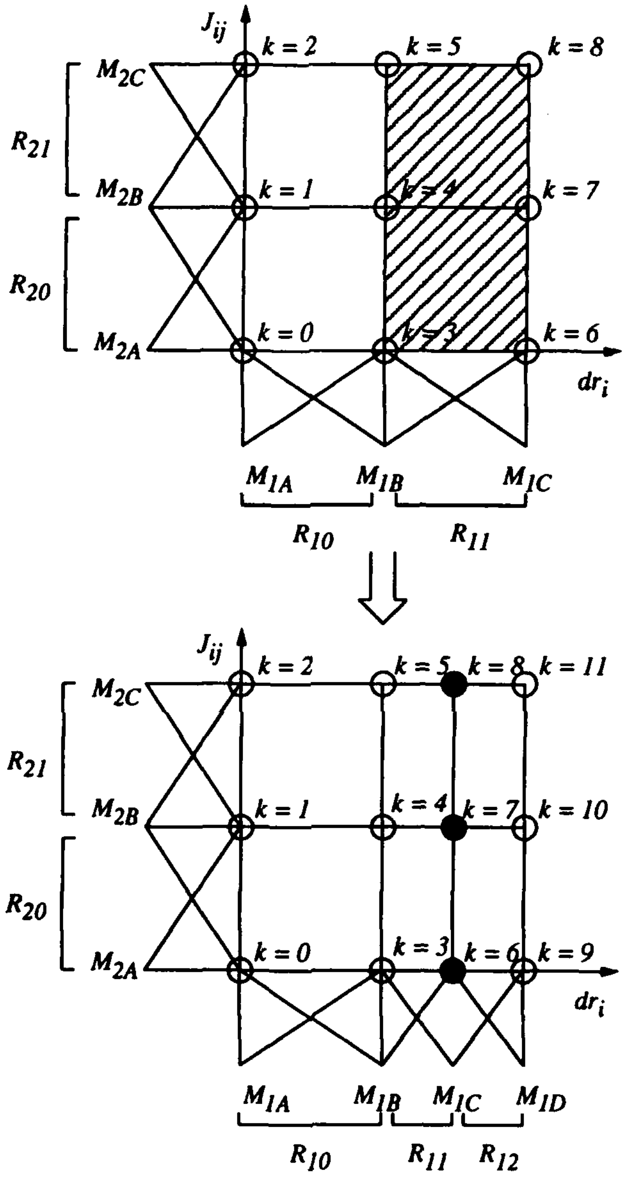


Figure 5: A new membership function is generated by splitting the region where the inference error is the highest. In the above diagram, the shaded region is assumed to have the largest error.

R_{max} will be divided into two equal halves as is shown in Figure 5 if both of the resulting regions contain at least one data vector in the input/output data table. If all input/output data is concentrated in one half of R_{max}, then, however, no reduction of e_{inf} would occur by splitting the region into two equal halves. In this case, the half of R_{max} that contains all the data points would be divided into equal halves so that R_{max} would be divided into two regions of 1/4 R_{max} and 3/4 R_{max}. Here, it must be verified again that both resulting regions contain at least one data point. Otherwise, the above procedure would be iterated again.

When a new membership function is generated for dr_i, then N_{Jij} new rules are created. Similarly, when a new membership function is generated for J_{ij}, then N_{dri} new rules are created.

The w_k for the updated rules corresponding to R^k will be weighted averages of the adjoining rules. In Figure 5, for example,

$$w_5^{new} = \frac{w_3 + w_6^{old}}{2} \qquad \text{(Eq. 23)}$$

$$w_6^{old} \rightarrow w_9 \qquad \text{(Eq. 24)}$$

In general, when the newly generated membership function $M_{1,m}$ is created for dr_i, then the updated w_k's will be,

$$\begin{aligned} w^{new}_{N_{Jij}(q-1)} &= \frac{1}{2}\left(w_{N_{Jij}(q-2)} + w^{old}_{N_{Jij}(q-1)}\right) \\ w^{old}_{N_{Jij}(q-1)} &\rightarrow w_{N_{Jij}q} \qquad q \in \{m, m+1, \ldots, m+N_{Jij}\} \end{aligned} \qquad \text{(Eq. 25)}$$

Similarly, when the newly generated membership function $M_{2,m}$ is created for J_{ij}, then the updated w_k's will be,

$$\begin{aligned} w^{new}_{qN_{dri}+m} &= \frac{1}{2}\left(w_{qN_{dri}+m-1} + w^{old}_{qN_{dri}+m}\right) \\ w^{old}_{qN_{dri}+m} &\rightarrow w_{qN_{dri}+m+1} \qquad q \in \{0, 1, \ldots, N_{Jij}-1\} \end{aligned} \qquad \text{(Eq. 26)}$$

Now, the input/output data must again be processed to adjust the w_k so as to reduce e_{inf}. This procedure is repeated until e_{inf} is reduced to 0.0013 for the fuzzy model output $d\theta_{ij}$

2.6 Scaling of Fuzzy Model Output

The values for $d\theta_j$, $j \in \{1, \ldots, n\}$, must be derived from the $6n$ $d\theta_{ij}$ terms. Define the following terms:

$$r_i = \sum_{j=1}^{n} |J_{ij}| \qquad \forall i \in \{1, \ldots, 6\} \qquad \text{(Eq. 27)}$$

$$c_j = \sum_{i=1}^{6} |J_{ij}| \qquad \forall j \in \{1, \ldots, n\} \qquad \text{(Eq. 28)}$$

Thus, r_i is the sum of the absolute values of the terms in the *ith* row of the Jacobian, and c_j is the sum of the absolute values of the terms in the *jth* column of the Jacobian. Now, scale each of the $d\theta_{ij}$ terms by row and column and form the effective joint angle by,

$$d\theta_j = \frac{1}{c_j}\left[\sum_{i=1}^{6} \frac{|J_{ij}|}{r_i} \times |J_{ij}| d\theta_{ij}\right] \qquad \text{(Eq. 29)}$$

This choice of scaling the individual $d\theta_{ij}$ ensures that (1) $d\theta_j$ will not be too large, and (2) the joint angles that can contribute the most to the motion in a given dr_i direction will in fact contribute the most [7].

2.7 Further Modifications

We further modify our scheme to improve performance in two ways: (1) Reduce the tracking error by introducing an adaptive fuzzy gain, and (2) Detect and suppress certain types of oscillations. The modifications themselves also apply fuzzy logic.

If the tracking error becomes too large, we would like the fuzzy controller to correct the problem quickly. This can be done by amplifying the error *dr* by a gain *K* so that the fuzzy model will take greater corrective measures than the error itself would prescribe. If, however, the error grows very large, there may be a potentially catastrophic problem and the fuzzy controller should proceed cautiously. Also, if oscillations are detected, this may be an indication that the gain *K* is too large and the fuzzy controller should once again proceed with greater caution.

Oscillations in our modified scheme are detected by monitoring sign changes in the $d\theta_j$, $j \in \{1, ..., n\}$. We monitor the previous ten values for each of the $d\theta_j$, and suspect oscillatory behavior when 4 or more sign changes occur.

Table I. Rule Base for Adaptive Gain/Oscillation Detection

K	*S*	*M*	*B*
OFF	1.00	2.60	0.80
ON	0.10	0.40	0.03

Figure 6 and Figure 7 show the membership functions we use to achieve an adaptive gain in order to control the tracking error and oscillations. Table I shows the rule base we use for the membership functions in Figure 6 and Figure 7.

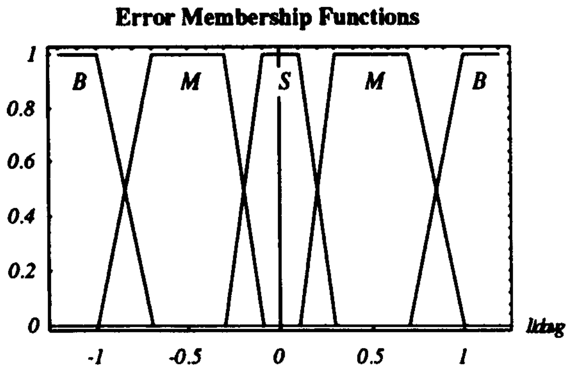


Figure 6: Membership functions used to select an adaptive gain based upon the average tracking error over the last 10 sample periods.

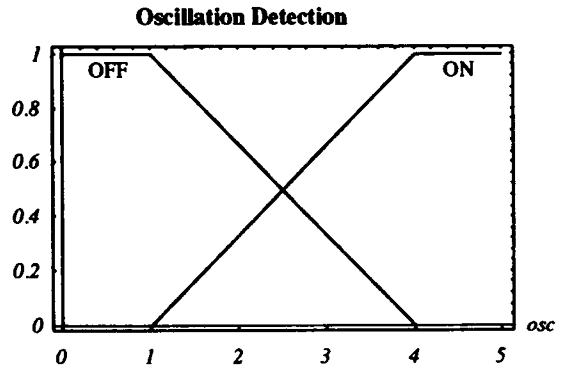


Figure 7: Membership functions for detecting oscillations. When 4 or more of the last 10 $d\theta_j$'s have changed sign, we consider oscillations to be "ON".

The membership functions in Figure 6 have a dynamic range based on the average error, d_{avg} for dr_i over the last ten control cycles. In other words, if the present error is greater than the averaged error over the last ten cycles, we consider the error big ("B"). All other errors are considered medium ("M") or small ("S"). Note from Table I that we suppress the gain K significantly when oscillations are detected.

Finally, we apply two low-pass filters to the input of the fuzzy model. First we apply low-pass filtering to the gain K,

$$K_{applied} = \frac{1}{3}K_{fuzzy} + \frac{2}{3}K_{applied,previous} \qquad \text{(Eq. 30)}$$

Second, we apply low-pass filtering to the resulting dr_i,

$$dr_i = \frac{1}{3}Kdr_i + \frac{2}{3}dr_{i,previous} \qquad \text{(Eq. 31)}$$

These two measures provide additional oscillation protection.

3 Fuzzy Model Characterisistics

The algorithm that we use to generate the fuzzy model for the $d\theta_{ij}$ produces 3 membership functions for dr_i and typically 15 membership functions for J_{ij}. Figure 8 shows the membership functions generated for dr_i, and Figure 9 shows the membership functions generated for a typical case of J_{ij}, where J_{ij} varies from -1.5 to 1.5.

Figure 10 shows a plot of the function that the fuzzy mapping was applied to, namely (Eq. 4). Since $d\theta_{ij}$ varies linearly with dr_i, e_{inf} could not be decreased by inserting more membership functions for dr_i. However, $d\theta_{ij}$ has a strong nonlinear dependence on J_{ij} near $J_{ij} = 0$ (Figure 10). Hence, that is where the most membership functions are generated, as can be seen in Figure 9.

Table II shows the rule base that was generated for the above case. A total of 15x3 = 45 rules were generated. Thus, the generated fuzzy model is relatively simple.

As was note previously, we can trivially extend the range of fuzzification for dr_i. Suppose that instead of assuming that dr_i varies from -0.1 to 0.1, we let dr_i vary from -0.2 to 0.2 in the fuzzy mapping. Then the w_k in the rule base are adjusted by multiplying each w_k, $k \in \{0, 1, \ldots, 44\}$, by 2. This can be done without loss of accuracy, since $d\theta_{ij}$ is linearly related to dr_i.

Table II.Rule Base Generated for $d\theta_{ij}$

dr_i \ J_{ij}	M_{2A}	M_{2B}	M_{2C}	M_{2D}	M_{2E}	M_{2F}	M_{2G}	M_{2H}	M_{2I}	M_{2J}	M_{2K}	M_{2L}	M_{2M}	M_{2N}	M_{2O}
M_{1A}	0.0667	0.1163	0.2502	0.4749	1.1398	1.5197	1.6174	-1.7134	-1.6174	-1.5197	-1.1398	-0.4749	-0.2502	-0.1163	-0.0667
M_{1B}	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
M_{1C}	-0.0667	-0.1163	-0.2502	-0.4749	-1.1398	-1.5197	-1.6174	1.7134	1.6174	1.5197	1.1398	0.4749	0.2502	0.1163	0.0667

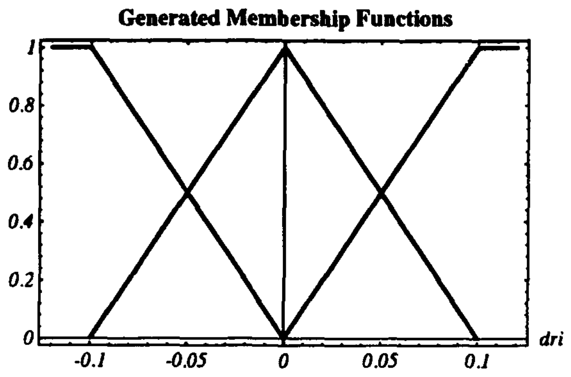


Figure 8: Membership functions generated for dr_i.

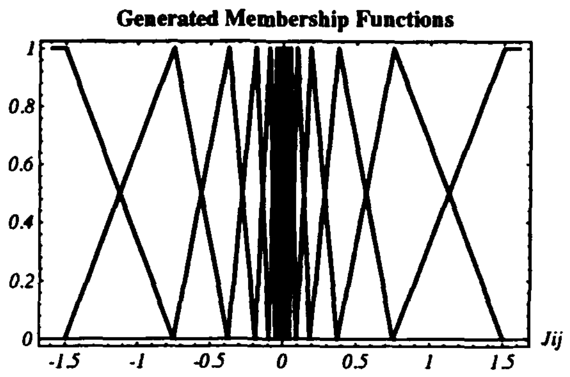


Figure 9: Membership functions generated for J_{ij}.

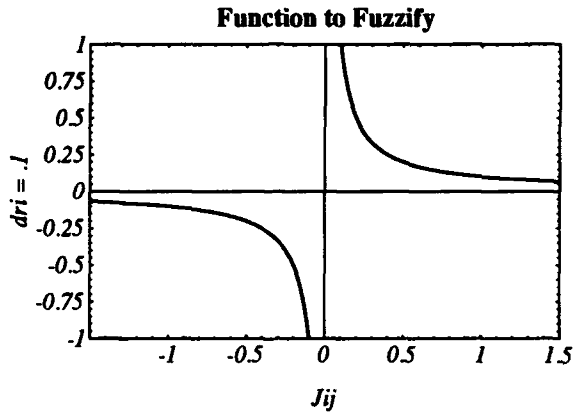


Figure 10: The nonlinear equation $d\theta_{ij} = \frac{dr_i}{J_{ij}}$ for $dr_i = 0.1$. Comparing Figure 10 to Figure 9 clearly shows that the most membership functions are generated at the region of highest nonlinearity.

4 Simulation Results

4.1 Efficient, Indirect Calculation of Fuzzy Model Output

The fuzzy model we have proposed thus far provides an intuitive basis for our approach and may be calculated *directly* with relative efficiency. As we demonstrate below however, (Eq. 32) leads to an alternate, *indirect* method of evaluating the fuzzy model which is roughly three times as efficient as *direct* evaluation of the fuzzy model.

Observe that in the scaling of the $d\theta_{ij}$ terms, each $d\theta_{ij}$ is multiplied by $|J_{ij}|^2$. Furthermore, note from (Eq. 4) that $d\theta_{ij} \approx \frac{dr_i}{J_{ij}}$ is the relation we originally fuzzified. We now propose to include the scaling multiplication of $|J_{ij}|^2$ in the fuzzy mapping so that,

$$d\hat{\theta}_{ij} \approx \frac{|J_{ij}|^2 dr_i}{J_{ij}} = \frac{J_{ij}{}^2 dr_i}{J_{ij}} = J_{ij} dr_i \qquad \text{(Eq. 32)}$$

$$d\theta_j = \frac{1}{c_j}\left[\sum_{i=1}^{6} \frac{d\hat{\theta}_{ij}}{r_i}\right] \qquad \forall i, r_i \neq 0, c_j \neq 0 \qquad \text{(Eq. 33)}$$

$$d\theta_j = 0 \qquad c_j = 0 \qquad \text{(Eq. 34)}$$

(Eq. 32) no longer degenerates for any value of J_{ij} an dr_i and is equivalent to calculating the fuzzy mapping *directly*. (Eq. 32), (Eq. 33), and (Eq. 34) provide an extremely efficient and robust algorithm for calculating the fuzzy inverse kinematic mapping of any redundant or nonredundant manipulator. The computational efficiency of (Eq. 32) through (Eq. 34) for calculating inverse kinematics will be evaluated and compared to other methods in a later section.

4.2 Simulation Implementation

We implemented the fuzzy model generator described in Section 3. For calculating $J(\Theta)$ and determining the range of each element J_{ij} in the Jacobian matrix, we use *Mathematica*, the symbolic math processing language. There, it is fairly straightforward to calculate (Eq. 6), (Eq. 7), and (Eq. 11). The remaining steps in Figure 2 are implemented in the *C* programming language, where the output from *Mathematica* is linked to *C* subroutines.

Then, we perform two different types of simulations. In the first case, we give as input to the fuzzy model only an initial value for Θ and a final desired position. We then let the manipulator move by repeatedly updating Θ so as to reach the final position. In the second case, we give as input to the fuzzy model an initial value for Θ, and a series of position data points that define the desired trajectory. Here, we update Θ once for every new data point. That is, the sampling frequency f_s is equal to the control frequency f_c. For the simulations presented in this paper, we assume,

$$f_s = f_c = 50 \text{ Hz} \qquad \text{(Eq. 35)}$$

All simulations were run using both the direct and the indirect method for calculating the fuzzy model. Results are nearly identical for both calculation schemes, where slightly smaller tracking errors and faster error convergence are observed for the indirect scheme. For the results presented below, the indirect, more efficient method of calculating the fuzzy model was used. All simulations were plotted in *Mathematica*.

Below, we present simulation results for one simple non-redundant manipulator, and two redundant manipulators. Tables III, IV, and V list the DH parameters for a planar, two-DOF manipulator, a planar, four-DOF manipulator, and a seven-DOF manipulator, respectively. These manipulators were used in the simulations presented below. All α_i and θ_i are in units of radians (rad), and all a_i and d_i are in units of meters (m).

Table III. DH Parameters for a 2-DOF Manipulator

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	1.0	0	θ_2
3	0	0.5	0	0

Table IV. DH Parameters for a 4-DOF, Planar Manipulator

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	0.50	0	θ_2
3	0	0.45	0	θ_3
4	0	0.35	0	θ_4
5	0	0.20	0	0

Table V. DH Parameters for a 7-DOF, 3D Manipulator

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	$\pi/2$	0	0.2	θ_2
3	$-\pi/2$	0	0.2	θ_3
4	$\pi/2$	0	0.2	θ_4
5	$-\pi/2$	0	0.2	θ_5
6	$\pi/2$	0	0.2	θ_6
7	$-\pi/2$	0	0.2	θ_7
8	0	0.2	0	0

4.3 Single, Large-Step Tracking

Below, we present results for the first type of simulation, which requires the fuzzy mapping to generate joint trajectories given only the initial and target position of the end-effector.

4.3.1 Two-DOF, Planar Manipulator:

Here two position coordinates (x, y) are mapped to two joint angles (θ_1, θ_2). Hence there are no redundant DOF's and the dimensions of $J(\Theta)$ are 2x2. When (θ_1, θ_2) = (0, 0), the end-effector is located at (1.5 m, 0.0 m).

For this simulation run, we ask the fuzzy mapping to generate the joint trajectories required to go from a position of (1.5 m, 0.0 m) to (0.0 m, 1.5 m). Note that both the initial and final positions specified are singular configurations for the manipulator. The simulation time is specified to be 1 sec.

Figure 11 shows the resulting trajectory generated by the fuzzy mapping. The manipulator converges to within 1 mm absolute error in 1 sec. Numerous other large-step trajectories were simulated with equal or better results. The steady-state position error always converges to zero.

4.3.2 Four-DOF, Planar Manipulator:

Here two position coordinates (x, y) are mapped to four joint angles $(\theta_1, \theta_2, \theta_3, \theta_4)$. Hence there are two redundant DOF's and the dimensions of $J(\Theta)$ are 2x4. When $(\theta_1, \theta_2, \theta_3, \theta_4)$ = (0, 0, 0, 0), the end-effector is located at (1.5 m, 0.0 m).

For this simulation run, we ask the fuzzy mapping to generate the joint trajectories required to go from a position of (1.5 m, 0.0 m) to (0.0 m, 1.0 m). The simulation time is specified to be 1 sec.

Figure 12 shows the resulting trajectory generated by the fuzzy mapping. The manipulator converges to within 0.1 mm absolute error in 1 sec. Numerous other large-step trajectories were simulated with equal or better results. For all attempted large-step trajectories, the steady-state position error converges to zero.

4.4 Multiple, Small-Step Tracking

Below, we present results for the second type of simulation, which requires the fuzzy controller to track a desired trajectory. In each case, the simulation runs for $t_1 + 0.4$ seconds, where t_1 denotes the duration of the trajectory, and 0.4 seconds is the steady-state time that we allow the manipulator to converge to the desired position. In all trajectory plots, the solid line represents the generated trajectory and the dotted line represents the desired trajectory.

4.4.1 Two-DOF, Planar Manipulator:

For this simulation, we require the manipulator to follow a curved pathwith the following characteristics:

$$l_{path} = 3.99m \qquad t_1 = 6s \qquad \bar{v} = 66\frac{cm}{s} \qquad \text{(Eq. 36)}$$

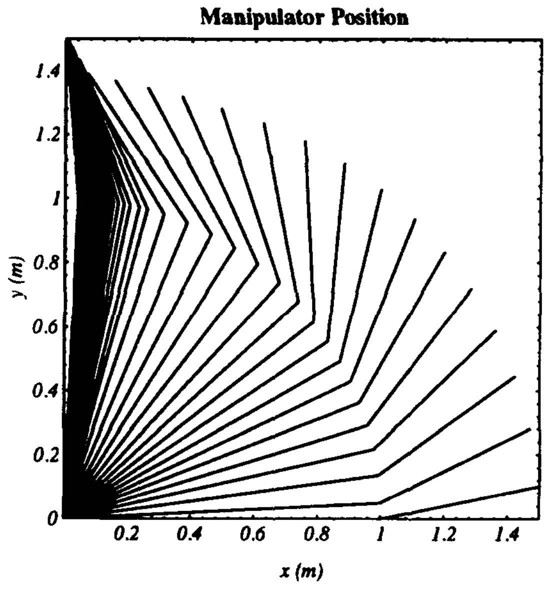


Figure 11: Trajectory generated to move from an initial position (1.5 m, 0.0 m) to a final position (0.0 m, 1.5 m) for a 2-DOF manipulator.

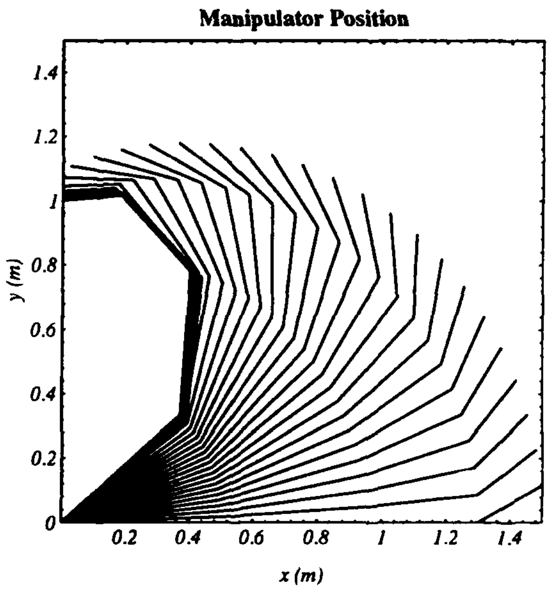


Figure 12: Trajectory generated to move from an initial position (1.5 m, 0.0 m) to a final position (0.0 m, 1.0 m) for a 4-DOF manipulator.

where l_{path} = length of the path, and $\bar{v}$ = average speed of the trajectory. The results of the simulation are shown in Figure 13, Figure 14, and Figure 15, with,

$$e_{max} = 2.27cm \qquad \bar{e} = 0.61cm \qquad e_{ss} = 0.00cm \qquad \text{(Eq. 37)}$$

where e_{max} = maximum deviation from the desired path, $\bar{e}$ = average deviation from the desired path, and e_{ss} = steady-state error after $t_1 + 0.4$. seconds.

As may be observed from Figure 13, the desired trajectory is tracked very closely by the generated trajectory. Part of the instantaneous error in Figure 14, therefore, is partially due to a small time lag. Also, note from Figure 15 that the generated joint paths appear to be smooth functions of time. We examine the joint paths more closely for the redundant manipulators.

4.4.2 Four-DOF, Planar Manipulator:

For this simulation, we require the manipulator to follow a curved path with the following characteristics:

$$l_{path} = 3.63m \qquad t_1 = 3s \qquad \bar{v} = 1.21\frac{m}{s} \qquad \text{(Eq. 38)}$$

The results of the simulation are shown in Figure 16, Figure 17, and Figure 18, with,

$$e_{max} = 3.49cm \qquad \bar{e} = 1.06cm \qquad e_{ss} = 0.00cm \qquad \text{(Eq. 39)}$$

Note from Figure 18, that the joint angle trajectories are smooth functions of time with little or no acceleration after the start and before the end of the motion.

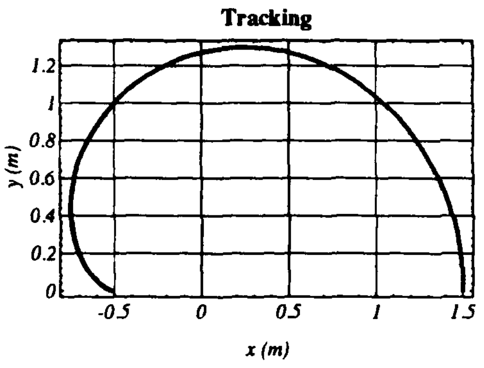


Figure 13: The desired and actual trajectories for the 2-DOF manipulator. Note that the two trajectories are so close that they completely overlap.

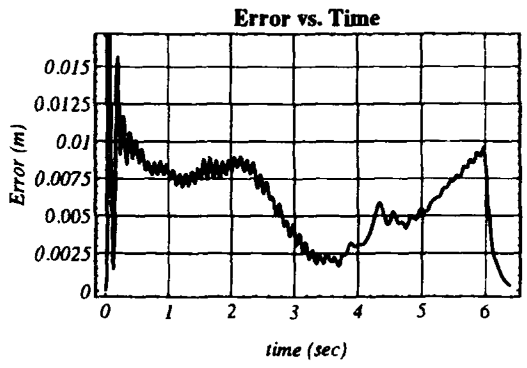


Figure 14: Instantaneous error between the desired path and the actual path generated for the 2-DOF manipulator. Note that the maximum error occurs early in the simulation.

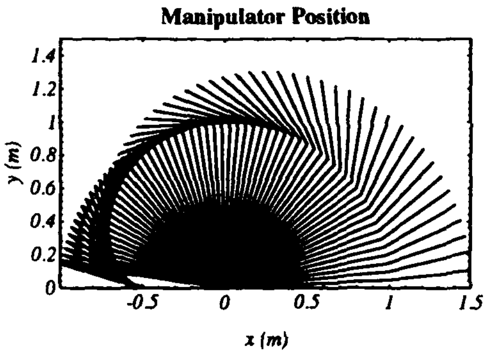


Figure 15: The generated trajectory of the fuzzy controller chosen to follow the desired trajectory.

4.4.3 Seven-DOF, Manipulator:

Here, we present simulations for position tracking of a seven-degree of freedom robot, whose DH parameters are given in Table V. Here three position coordinates (x, y, z) are mapped to seven joint angles $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7)$. Hence, there are four redundant DOF's and the dimensions of $J(\Theta)$ are 3x7. When $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7) = (0, 0, 0, 0, 0, 0, 0)$, the end-effector is located at (0.2 m, -0.6 m, 0.6 m). We chose not to include orientation tracking for the end-effector for simplicity and in order to demonstrate tracking for a hyper-redundant manipulator.

For the first simulation, we require the manipulator to follow a complex path with the following characteristics:

$$l_{path} = 5.28m \qquad t_1 = 12.6s \qquad \bar{v} = 42\frac{cm}{s} \tag{Eq. 40}$$

The results of the simulation are shown in Figure 19 and Figure 20, with,

$$e_{max} = 1.98cm \qquad \bar{e} = 0.70cm \qquad e_{ss} = 0.08cm: \tag{Eq. 41}$$

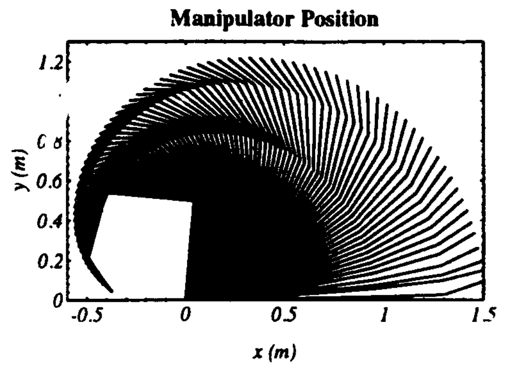


Figure 16: This is the path that the fuzzy controller chooses for the 4-DOF manipulator.

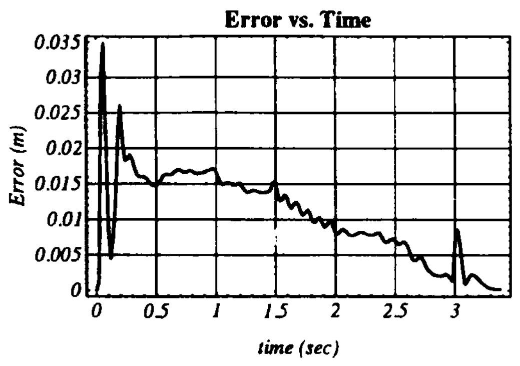


Figure 17: Instantaneous error between the actual and desired paths for the 4-DOF manipulator.

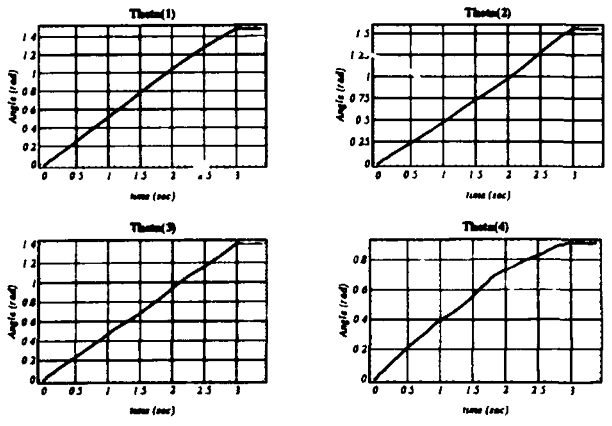
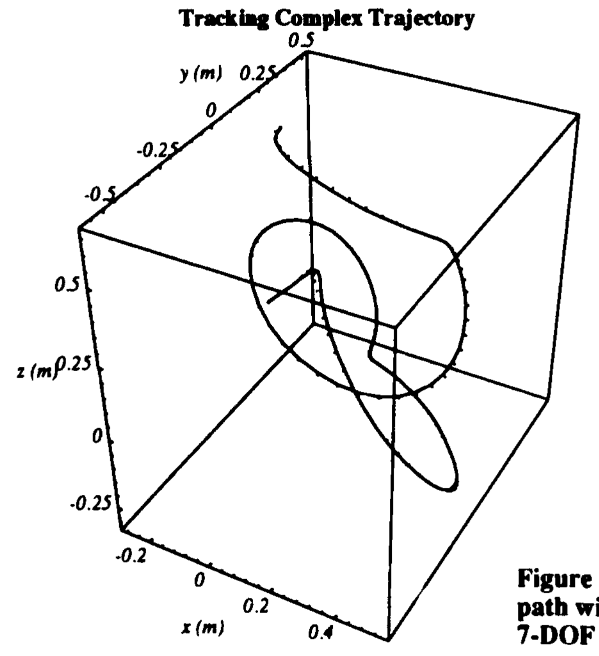


Figure 18: The joint angle trajectories for the 4-DOF manipulator are all smooth, near constant velocity trajectories.



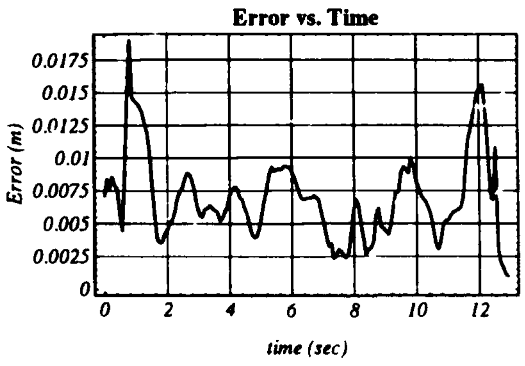


Figure 20: Instantaneous error between the desired path and the generated trajectory for the first simulation of the 7-DOF manipulator.

Figure 19: The generated trajectory tracks the desired path with good accuracy even through sharp turns for the 7-DOF manipulator.

For the second simulation, we require the manipulator to follow a curved path with the following characteristics,

$$l_{path} = 31.55cm \qquad t_1 = 4.0s \qquad \bar{v} = 7.90\frac{cm}{s} \tag{Eq. 42}$$

The results of the simulation are shown in Figure 21 and Figure 22, with,

$$e_{max} = 0.71cm \qquad \bar{e} = 0.45cm \qquad e_{ss} = 0.09cm$$

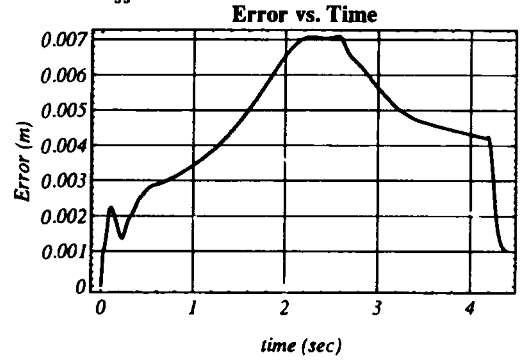


Figure 22: Instantaneous error between the desired path and the generated trajectory for the curved trajectory in 3-space.

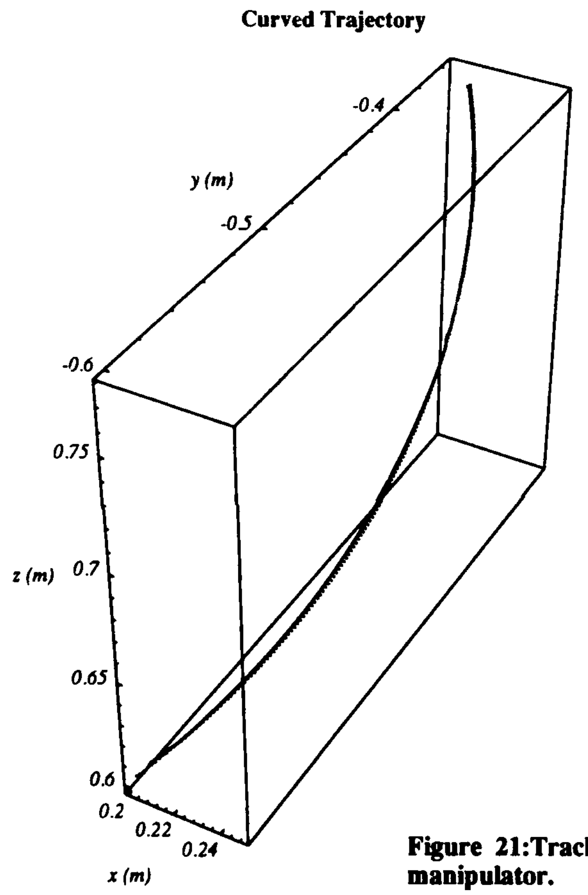


Figure 21:Tracking of a curved trajectory in 3-space for a 7-DOF manipulator.

For the last simulation of the 7-DOF manipulator, we require the manipulator to follow a straight path with starting coordinates of (0.2, -0.6, 0.6) and final coordinates of (0.7, -0.35, -0.15) and the following characteristics:

$$l_{path} = 93.5cm \qquad t_1 = 10.0s \qquad \bar{v} = 9.35\frac{cm}{s} \tag{Eq. 43}$$

The resulting joint trajectories are shown in Figure 23. The various tracking errors for the trajectory are:

$$e_{max} = 1.10cm \qquad \bar{e} = 0.59cm \qquad e_{ss} = 0.09cm \tag{Eq. 44}$$

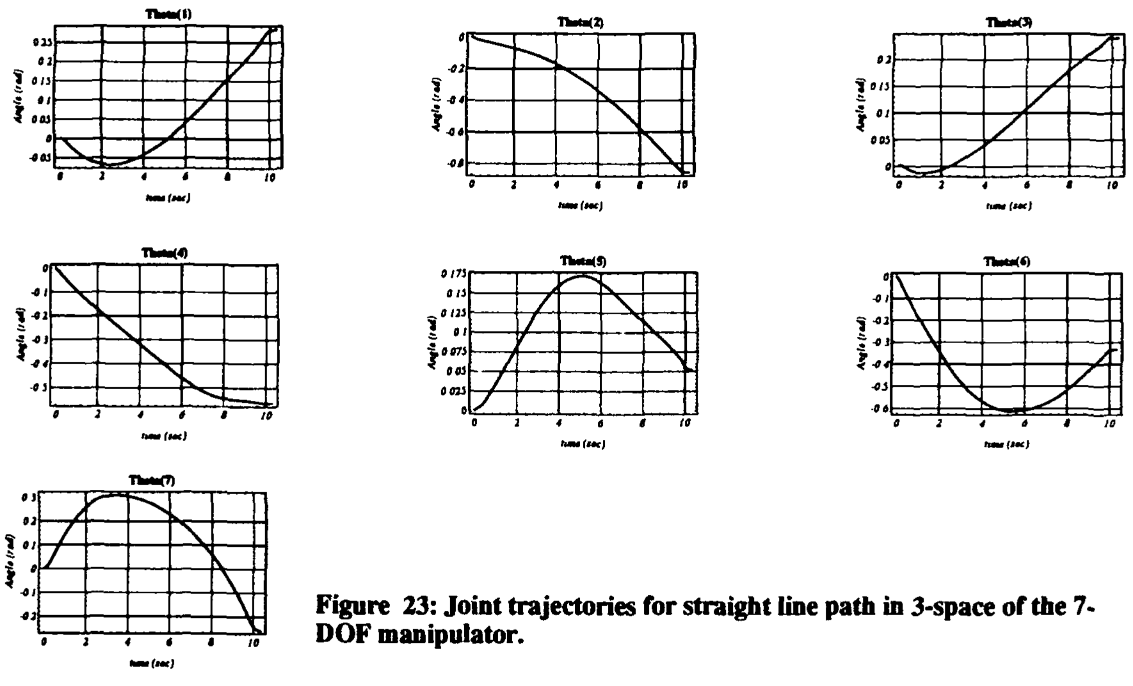


Figure 23: Joint trajectories for straight line path in 3-space of the 7-DOF manipulator.

5 Discussion

5.1 Comments on Simulation Results

Here, we note some of the main characteristics about the fuzzy controller performance. First and most important, the fuzzy controller produces *zero* steady-state error. In the simulation results presented in Section 4, all simulations converged to within 1 mm of the desired position in 0.4 seconds after the desired trajectory had stopped changing. In all instances, the steady-state error converged to zero in less than 1 second.

The average tracking error is a function of the average speed of the end-effector during tracking and the complexity of the trajectory. The highest average tracking error of 1.06 cm occurred in the simulation of the four-DOF planar manipulator, where the average trajectory speed was 1.21 m/s. The lowest average tracking error of 0.45 cm occurred in the simulation of the complex trajectory for the seven-DOF robot. Here the average speed of the end-effector was only 7.90 cm/s.

To get a better understanding of the relationship between the average speed of the end-effector and the tracking error, we simulated the straight-line trajectory for the seven-DOF manipulator at various speeds. Table VI reports the results. Note that although the speed is increased by 400% from 9.35 cm/s to 46.77 cm/s, the average tracking error increases by only 78% from 0.59cm to 1.05 cm. The maximum tracking error increases by 279% from 1.10 cm/s to 3.07 cm/s. Also note that the average tracking error for a speed of 46.77 cm/s for the seven-DOF manipulator is roughly equivalent to the tracking error for the four-DOF planar manipulator at a speed of 1.21 m/s. This is a consequence of the fact that straight-line paths are, in general, more difficult to track than curved paths.

Table VI. Relationship Between Speed and Tracking Error

v_{avg} (f_s = 50 Hz)	e_{max}	e_{avg}
9.35 cm/sec	1.10 cm	0.59 cm
18.71 cm/sec	1.57 cm	0.82 cm
46.77 cm/sec	3.07 cm	1.05 cm

The maximum error during tracking occurs in general (but no always) near the beginning of the trajectory, when the manipulator is still adjusting the joint angles to track the generated path more easily. Furthermore, the maximum error does not necessarily represent the maximum deviation from the desired path, but may, at least in part, reflect some time lag.

Note that increasing the speed of the end-effector is equivalent to reducing the control frequency f_c and vic versa. Therefore, the above discussion applies equally well to variations in the control frequency.

Second, we examine the actual joint trajectories generated by the fuzzy controller. Figure 18 and Figure 23 show the joint angle trajectories generated for the four-DOF and seven-DOF simulations respectively. Note that in both instances, the joint trajectories generated by the fuzzy controller are smooth, non-oscillatory functions of time.

For the four-DOF simulation, we see that the generated trajectories vary nearly linearly with time. This means that the manipulator will move at near-constant velocities and little or no acceleration. For the seven-DOF straight-line simulation, we see that the joint trajectories no longer vary linearly with time but are still smooth functions of time. Also, the change in slope of the joint trajectories (i.e. the acceleration of the joints) increases with joints that are further away from the base. The joint trajectories for θ_5, θ_6, and θ_7 exhibit much higher accelerations than do the trajectories for θ_1, θ_2, θ_3, and θ_4. When given a choice, the fuzzy controller seems to prefer moving links closer to the end-effector over links that are closer to the base. This is a desirable characteristic in that links that are closer to the end-effector require less torque to move and are easier to control.

Third, the simulations of single, large-step tracking indicate that the fuzzy controller is able to converge quickly to a desired position even when the initial error is very large. If the speed of the desired trajectory should suddenly change, the fuzzy controller will still be able to overcome any large error without much problem.

Fourth, we note that the simulations were performed near or at singularities at various points in the trajectory. In all cases, the fuzzy controller proved robust and handled the singularities without much difficulty.

Finally, when the modifications to the fuzzy scheme in Section 2.7 are removed, we observe significant increases in the maximum error, mean error, and oscillations. Schacherbauer and Xu [7] treat this topic in significant detail for similar input modifications to another fuzzy inverse kinematic scheme. Their results show that the low-pass filtering applied at the input of the fuzzy controller contributes the most to reduction in error.

5.2 Computational Efficiency Analysis

In order to demonstrate the usefulness of our scheme for real-time control, we analyze the computational efficiency of our method for calculating the inverse kinematics of an n-DOF manipulator. For the development below we restrict the twist angle α to $0°$ and $\pm 90°$.

In each control cycle, $J(\Theta)$ must be evaluated. This calculation will, in general, require at most ($30n$ - 55) multiplications, ($15n$ - 38) additions and ($2n$ - 2) sine/cosine evaluations [6] for both position and orientation tracking. Here $n \geq 3$.

We now calculate the number of arithmetic operations required to calculate the inverse kinematics once the Jacobian has been determined. From (Eq. 32), one multiplication is required for each J_{ij} term in $J(\Theta)$ to calculate the $d\hat{\theta}_{ij}$'s. Hence, to calculate all the $d\hat{\theta}_{ij}$, we require $6n$ multiplications. From (Eq. 27) and (Eq. 28), a total of ($6n$ - 6) additions are required to calculate the r_i terms, and $5n$ additions are required to calculate the c_j terms. From (Eq. 33), a further $6n$ divisions are required per J_{ij}. To form the $d\theta_j$ terms, an additional n divisions and $5n$ additions are required. (Eq. 32) through (Eq. 34) therefore require,

$$6n + 6n + n = 13n \text{ multiplications/divisions}$$

$$5n + 5n + 6n - 6 = 16n - 6 \text{ additions/subtractions} \qquad \text{(Eq. 45)}$$

Including the Jacobian calculations, we require a total of,

$43n$ - 55 multiplications/divisions
$31n$ - 44 additions/subtractions (Eq. 46)
$2n$ - 2 sine/cosine function evaluations

Therefore, the computational complexity of this scheme increases only linearly with n. For a general 6-DOF manipulator where *none* of the link distance parameters are assumed to be zero, and the twist angles α are assumed to be either 0° or ±90°, no closed-form solution exists for the inverse kinematics of such a manipulator. Our scheme, however, would require only 203 multiplications/divisions, 142 additions/subtractions, and 10 sine/cosine evaluations per control cycle. Furthermore, under the same assumptions, for a general 7-DOF redundant manipulator, we require only 246 multiplications/divisions, 173 additions/subtractions, and 12 sine/cosine evaluations per control cycle.

The Puma 560 manipulator is kinematically very simple in that only the link lengths a_2, and a_3, and the link offsets d_3 and d_4 are nonzero [5]. Because of its kinematic simplicity, a closed form solution for the inverse kinematics exists. Paul and Zhang [5] determined the minimum number of arithmetic operations involved in solving the inverse kinematics of the Puma 560. Given the transformation ${}^0T_6$, the required arithmetic operations are,

37 multiplications/divisions
22 additions/subtractions
2 square root function evaluations (Eq. 47)
6 arctan function evaluations
2 arcsin/arccos function evaluations
4 sine/cosine function evaluations

However, we assume that only the desired position/orientation vector r is given. In that case, additional arithmetic operations are required to evaluate the ${}^0R_6$ rotation submatrix of ${}^0T_6$. In terms of the fixed Euler rotation angles, ${}^0R_6$ is given by,

$${}^0R_6 = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix} \qquad \text{(Eq. 48)}$$

This will require,

16 multiplications
4 additions (Eq. 49)
6 sine/cosine function evaluations

In total, the closed-form solution will require,

53 multiplications/divisions
26 additions/subtractions
2 square root function evaluations (Eq. 50)

6 arctan function evaluations
2 arcsin/arccos function evaluations
10 sine/cosine function evaluations

to calculate the inverse kinematics. The Jacobian for the Puma 560 has 15 entries that are equal to 0 [5],

$$J(\Theta) = \begin{bmatrix} J_{11} & J_{12} & J_{13} & 0 & 0 & 0 \\ J_{21} & J_{22} & J_{23} & 0 & 0 & 0 \\ J_{31} & J_{32} & J_{33} & 0 & 0 & 0 \\ J_{41} & 0 & J_{43} & J_{44} & J_{45} & 0 \\ J_{51} & 0 & J_{53} & J_{54} & J_{55} & 0 \\ J_{61} & 0 & J_{63} & J_{64} & 0 & 1 \end{bmatrix} \qquad \text{(Eq. 51)}$$

The evaluation of $J(\Theta)$ requires,

46 multiplications
19 additions/subtractions (Eq. 52)
6 sine/cosine function evaluations [5]

The number of calculations required for the inverse kinematics will be only a fraction of the number of calculations required for a general 6-DOF manipulator since no calculations are required for the 15 zero terms. To calculate the $d\hat{\theta}_{ij}$ for the nonzero J_{ij} terms, we require (36-15) = 21 multiplications. Furthermore, 15 additions are required to calculate the r_i terms, and 15 additions are required to calculate the c_j terms. The scaling of (Eq. 32) requires an additional [(36-15)+5] = 26 multiplications/divisions and a further 15 additions. Calculating the inverse kinematics, therefore requires,

47 multiplications (Eq. 53)
45 additions/subtractions

for the Puma 560. Including the Jacobian calculations, we require,

93 multiplications/divisions
64 additions/subtractions (Eq. 54)
6 sine/cosine function evaluations

To compare the calculations required for the closed-form solution and our fuzzy model approach more directly, we will make the following assignments for each arithmetic operation in terms of "units of computing power required,"

1 multiplication = 1 unit
1 addition/subtraction = 1/3 units
1 sine/cosine evaluation = 5 units (Eq. 55)
1 inverse function evaluation = 7 units
1 square root evaluation = 4 units [2]

The first two entries in Table VII show that even for the case when a closed-form solution exists for the inverse kinematics of a manipulator, our proposed method is marginally more efficient. Furthermore, the Puma 560 is a best-case scenario for conventional inverse kinematics. For only a very few kinematically simple manipulators will the results be as good in terms of computational efficiency. The efficiency of inverse kinematics rapidly degenerates when more than a few of the distance link parameters are nonzero. In fact, the most general manipulator for which a closed-form solution does exist is a six-DOF robot where the last three DOFs affect only the orientation of the end-effector, as is the case with the Puma 560 [2].

Below, we compare the computational efficiency of our proposed method to the most efficient solution for the inverse kinematics of a redundant manipulator presented by Nakamura [4]. Since both methods require calculation of $J(\Theta)$, we only compare the additional arithmetic operations required to calculate the inverse kinematics once $J(\Theta)$ is calculated.

Nakamura presents an inverse kinematic solution for the case of redundant manipulators using the pseudo-inverse of the Jacobian matrix. As a function of n, this method requires,

$(33n + 112)$ multiplications/divisions (Eq. 56)
$(33n + 64)$ additions/subtractions

In comparison, our method requires,

$13n$ multiplications/divisions (Eq. 57)
$(16n-6)$ additions/subtractions

Therefore, our method is roughly two and a half times more efficient than the pseudo-inverse method. Table VII compares the total number of arithmetic operations required for a general 7-DOF and 8-DOF manipulator. Again, the fuzzy method is significantly more efficient than the pseudo-inverse method.

Table VII. Computational Efficiency

	Mult./Div	Add./Sub.	Sin/Cos	Inv. Func.	Sqrt.	"UCP"[4]
P560[1]	53	26	10	8	2	$\boxed{176}$
P560[2]	93	64	6	0	0	$\boxed{144}$
7DOF[3]	498	362	12	0	0	$\boxed{679}$
7DOF[2]	246	173	12	0	0	$\boxed{364}$
8DOF[3]	561	410	14	0	0	$\boxed{768}$
8DOF[2]	289	204	14	0	0	$\boxed{427}$

[1]Closed-form inverse kinematics
[2]Fuzzy inverse kinematic mapping
[3]Inverse kinematics through pseudo-inverse of Jacobian
[4]UCP = Units of Computing Power

6 Conclusion

Calculating exact inverse kinematics in real-time is computationally too burdensome for all but the most simple kinematic configurations. For a wide class of robot tasks such as teleoperation, however, we do not require exact inverse kinematics during global positioning and trajectory following. Here, we have presented a method of calculating inverse kinematics which has been shown to be robust to singular configurations, and is applicable to both redundant and nonredundant manipulators. The inverse kinematic mapping proposed trades off small tracking error for computational efficiency and robustness, and allows robot redundancy to be exploited rather than averted. The fuzzy method is much more efficient for redundant manipulators than other currently available methods and has been shown to be marginally more efficient even for a simple robot where a closed-form solution to the inverse kinematic problem exists. Furthermore, the method converges quickly in steady state and produces zero steady state error in position and orientation of the end-effector.

7 Acknowledgement

We would like to thank David Simon, whose software on fuzzy model generation aided in the development of the software written for the simulations in this paper.

References

[1] S. Araki, H. Nomura, I. Hayashi and N. Wakami, "A Self-Generating Method of Fuzzy Inference Rules," in *Proceedings: 1991 IFES Conference*, pp. 1047-1058, 1991.

[2] J. J. Craig, *Introduction to Robotics: Mechanics and Control, 2nd ed.* New York, Addison-Wesley Publishing Company, 1989.

[3] C. C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller, Parts I and II," in *IEEE Transactions on Systems, Man and Cybernetics*, vol. 20, no. 2, pp. 404-435, 1990.

[4] Nakamura, Yoshihiko, *Advanced Robotics: Redundancy and Optimization*, New York, Addison-Wesley Publishing Company, 1991.

[5] R. P. Paul and Hong Zhang, "Computationally Efficient Kinematics for Manipulators with Spherical Wrists Based on the Homogeneous Transformation Representation," in *The International Journal of Robotics Research*, vol. 5, no.2, pp. 32-44, 1986.

[6] D. E. Orin and W.W. Schrader, "Efficient Jacobian Determination for Robot Manipulators," in *Robotics Research: The First International Symposium*, M. Brady and R. P. Paul, Editors, MIT Press, 1984.

[7] A. Schacherbauer and Y. Xu, "Fuzzy Control and Fuzzy Kinematic Mapping for a Redundant Space Robot," Technical Paper, CMU-RI-TR-92-12, Carnegie Mellon University, 1992.